Take a look at the exam question below and the fourteen answers provided by students. All but one are wrong. Find and indicate the problem with each incorrect solution, and circle the correct solution.

1. You have been hired to coordinate a dancefloor. Your task is to match dancers from two teams, team red and team blue, into pairs. Each dancer is a separate thread that executes the following code:

```
red() {
      redTeamMemberReady();
      // must not get here unless there is a corresponding
      // member of the blue team on the dancefloor
      tango();
}

blue() {
      blueTeamMemberReady();
      // must not get here unless there is a corresponding
      // member of the blue team on the dancefloor
      tango();
}
```

A team may have an arbitrary number of members, and the two teams need not have the same number of members. Your task is to coordinate the dancefloor by writing the code for the two procedures `redTeamMemberReady()` and `blueTeamMemberReady()`. You need to ensure that your solution never lets unpaired dancers onto the dance floor, and does not unnecessarily keep dancers waiting when it is possible to make progress. You need not ensure that all dancers are treated fairly; that is, it is fine to let a late arriving dancer onto the dancefloor ahead of other dancers who have been waiting for some time.

Synchronize the dancers using monitors with condition variables. Use the signal-delayed discipline (Mesa-style condition variables).

1. Monitor Mon {
    CondVar BlueReady = 0, RedReady = 0;

    redTeamMemberReady(){
        BlueReady.signal();
        RedReady.wait();
    }
    blueTeamMemberReady(){
        RedReady.signal();
        BlueReady.wait();
    }
}

```
2. Monitor m {
        int bluecount = 0, redcount = 0;
        CondVar red, blue;

        RedTeamMemberReady() {
                if (redcount > 0 && bluecount > 0) {
                        bluecount--;
                        blue.broadcast();
                } else {
                        redcount++;
                        conditions      red wait
                }
        }
        BlueTeamMemReady() {
                if (bluecount>0 && redcount>0){
                        redcount--;
                        red.broadcast();
                } else {
                        bluecount++;
                        condition       blue wait
                }
        }
}
```

3.

```
Monitor m {
        int redcount=0, bluecount=0;
        CondVar r,b;

        redTeamMemberReady(){
                if((redcount != (bluecount – 1)) && (redcount != bluecount))
                        r.wait();
                redcount++;
                b.signal();

        }

        blueTeamMemberReady(){
                if((bluecount !=(redcount – 1)) && (bluecount != redcount))
                        b.wait();
                bluecount++;
                r.signal();
        }
}
```

4. Monitor m{

```
CondVar RedCondition, BlueCondition, PartnerCondition;
boolean someRedReady = false, someBlueReady = false;

redTeamMemberReady()      {
        while (someRedReady){
                RedCondition.wait();

        someRedReady = true

        while (!partnerReady){
                partnerReady = true
                partnerCondition.wait()
        }
        partnerCondition.signal()

        someRedReady = false
}
```

For blueTeamMemberReady() replace “Red” with “Blue” in above code.

}

5.

```
Monitor m {
        CondVar         red, blue, partner;

        int numberRedWaiting=0, numberBlueWaiting=0;
        bool partnerWait = false;

        void redTeamMemberReady(){
                if(partnerWaiting)
                  partner.wait();
                if (numberBlueWaiting >0)
                    numberBluewaiting--;
                    blue.signal();
                    partnerWaiting=true;
                } else {
                    numberRedWaiting++;
                    red.wait();
                    partnerWaiting=false
                    partner.signal();
                }
        }
```

For blueTeamMemberReady() replace “red” with “blue” in above code and “blue” with “red”.

```
}
```

6. Monitor m {

```
Condition needRed, needBlue;
Int redCount=0; blueCount=0;

RedTeamMemberReady(){
        while (bluecount==0){
                redCount++;
                needBlue.wait();
                redcount--;
        }
        need Red.signal();
}

blueTeamMemberReady() {
        while (redcount==0){
                blueCount++:
                needRed.wait():
                blueCount--;
        }
        need Blue.signal();
}
```

}

7. Monitor m {

```
int Rwaiting [n], Bwaiting[n];
CondVar R[n], B[n];

blueTeamMemberReady() {
        for (i=0, i<n, ++i) {
                if (R waiting [i]==1){
                        R[i].signal()
                        Rwaiting[i]=0
                        return;
                }
        }
        for (i=0, i<n, ++i) {
                if (Bwaiting [i]==0) {
                        Bwaiting[i]=1,
                        B[i].wait();
                        break;
                }
        }
}
redTeamMemberReady() {
        for (i=0, i<n, ++i) {
                if (Bwaiting [i]==1){
                        B[i].signal();
                        Bwaiting[i]=0;
                        return;
                }
        for (i=0, i<n, ++i) {
                if (Rwaiting [i]==0) {
                        Rwaiting[i] = 1;
                        R[i].wait();
                        break;
                }
        }
}
Initialize () {
        for (i=0, i<n, ++i)  {
                Rwaiting[i]=0;
                Bwaiting[i]=0;
        }
}
```

}

8. Monitor m {

```
    CondVar redwaiting, bluewaiting;
    int rwcount=0, bwcount=0;
    redTeamMemberReady()
    {
        if (bwcount >= 0)
        {
            bwcount--;
            bluewaiting.signal();
        } else {
            rwcount++;
            redwaiting.wait();
        }
    }
    blueTeamMemberReady()
    {
        if (rwcount>=0)
        {
            rwcount--;
            redwaiting.signal();
        } else {
            bwcount++;
            bluewaiting.wait();
        }
    }
}
```

9. Monitor m {

```
    CondVar     waitingForBlue, waitingForRed;
    int         redCount=0, blueCount=0;

    redTeamMemberReady(){
        if (blueCount==0){
            redCount++;
            signal(waitingForBlue);
        } else {
            blueCount--;
            signal (waitingForRed);
        }
    }

    blueTeamMemberReady(){
        if (redCount==0){
            blueCount++;
            wait (waitingForRed);
        } else {
            redCount--;
            signal (waitingForBlue);
        }
    }
}
```

10.

```
Monitor m {
int       blueWaiting, redWaiting;
CondVar         red, blue;

          redTeamMemberReady() {
               if(blueWaiting >0) {
                    blue waiting---;
                    signal(blue);
               } else {
                    redWaiting++;
                    wait(red);
               }
          }

          blueTeamMemberReady() {
               if(redWaiting >0) {
                    redWaiting--;
                    signal(red);
               } else        {
                    blueWaiting++;
                    wait(blue);
          }
}
```

11. Monitor m {

```
        int       redcount=0, bluecount=0
        CondVar         red, blue.

        redTeamMemberReady() {
             if (bluecount<=0) {
                  blue.signal();
                  bluecount++;
             }
             redcount--;
             if(redcount<0);
                  red.wait();
        }
        blueTeamMemberReady() {
             if (redcount<=0) {
                  red.signal();
                  redcount++;
             }
             bluecount--;
             if (bluecount<0)
                  blue.wait();
        }
}
```

12. Monitor m {

```
        CondVar         red, blue;
        int         rWaiting=0, rDancing=0,
        int         bWaiting=0, bDancing=0,

        redTeamMemberReady() {
            if (bWaiting>0) {
                rDancing++;
                red.signal(),
                return;
            }
            rWaiting++;
            while       (bDancing <= rDancing) {
                blue,wait();
            }
            rWaiting--;
            rDancing++;
            return;
        }

        blueTeamMemberReady() {
              if  (rWaiting>0) {
                bDancing++;
                blue.signal;
                return;
             }
             bWaiting++;
             while (rDancing <= bDancing) {
                red.wait();
            }
            bWaiting--;
            bDancing++
            return,
        }
}
```

13.

```
Monitor m {
        int redwaiting=0, bluewaiting=0;
        CondVar cv;

        redTeamMemberReady(){
                ++redwaiting;
                if (bluewaiting<=0){
                        cv.wait();
                }
                --redwaiting;
                cv.signal();
        }

        blueTeamMemberReady(){
                ++bluewaiting;
                if (readwaiting<=0){
                cv     wait();
                }

                --bluewaiting;
                cv.signal();
        }
```

14.

```
Monitor m      {
        CondVar        new_red, new_blue;
        int       red_free = 0, blue_free=0;

        redTeamMemberReady() {
                red_free++;
                signal (new_red);
                while (blue_free==0)
                        wait (new_blue);
                blue_free--;
        }

        blueTeamMemberReady() {
                blue_free++;
                signal (new_blue);
                while (red_free==0)
                        wait (new_red);
                red_free--;
        }
}
```

15. Solve problems 8.13, 9.13, 9.16, and 10.11 (for 10.11, only consider the cases where two, three and four frames are available).